MICROBLINK
See beyond.

Kotlin and The Cool Features We Used

Patrik Durasek

JavaCro'21

Roadmap

- Who am I?
- Why Kotlin?
- The Features
- The Issues

Who Am I

- Patrik Durasek
- Graduated from RIT Croatia
- 4 Years in the Industry

Microblink

- Microblink is an AI company that develops computer vision technology that makes life easier for more than 100 million end-users across the world
- BlinkID is our flagship product
  - BlinkID is an ID scanning software, and it's used in a number of use cases, including bank account opening, identity verification and visitor management
  - Users can either scan their document in real time from camera feed or upload its image from gallery and have the results extracted after a couple of seconds
- We are currently developing a complete identity verification solution

Why Kotlin?

- Less code
- Concise and clear
- Syntactic sugar

The Features

- Data classes
- Null safety
- Default parameters
- Extension Functions
- “Streamless” functions
- Assigning values with conditionals
- Return If
- Coroutines

Data Classes

- Reduce the amount of code
- Concise

```
data class IdBarcodeRequest(
   val image: ImageSource?,
    val inputString: String?,
    val ageLimit: Int?
)
```

Null Safety

- Variables not nullable by default
- Have to explicitly declare a nullable variable
- Out of the box null checks

Default parameters

- Allows default argument values
- Named Parameters
- Allows named parameters in constructors

```
return BlinkIdEndpointResponse(
    traceId = request.traceId,
    executionId = request.executionId,
    data = result,
    startTime = startTime,
    finishTime = Instant.now()
)
```

Extension Functions

- Change functions without changing the source code
- Change functionality without having to inherit from the class or use design patterns
- Write new functions for a class from a third-party library that you can't modify

```kotlin
fun ByteArray.startsWith(compareTo: ByteArray): Boolean {
    var i = 0
    while (i < compareTo.size) {
        if (this[i] != compareTo[i]) {
            return false
        }
        i++
    }
    return true
}
```

“Streamless" collections

- Collections API
- Filter, map, flatmap, etc.

```
val allowedProtocols = inputAllowedProtocols.filter { !it.isNullOrBlank() }
```

Assigning Values with Conditionals

- "If" is a an expression, not a statement
- Expressions return a value
- Values can be directly assigned to variables

```
Int min = x
if (y < x) {
    min = y
}
```

→

```
val min = if (x < y) x else y
```

Return If

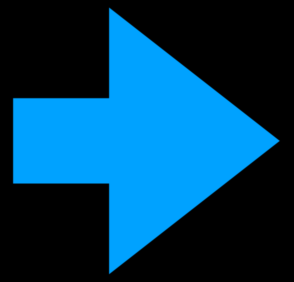
- Expressions can be returned
- Same applies to "try"

```
return if (type.isCompressed()) {
    ProcessedCompressedImage(base64ToByteArray(sanitizedBase64String), imageMetadataBuilder.build())
} else {
    throw IllegalStateException("Unhandled image type")
}
```

Coroutines

- Write asynchronous or non-blocking code
- Managed by the user
- Can be suspended

```
fun sendRequest(): Int {
    /* do some heavy work */
    return 1;
}
```



```
suspend fun sendRequest(): Int {
    /* do some heavy work */
    return 1;
}
```

Coroutines Cont'd

- Dispatchers
  - Default
  - IO
  - Unconfined

```kotlin
open fun processImage(frontImage: UnprocessedImage, backImage: UnprocessedImage?):
ProcessedImagePair {
    log.debug("Processing image pair")
    return runBlocking(Dispatchers.IO + MDCContext()) {
        val jobFrontImage =
            async {
                imageProcessingService.processSingleImage(
                    frontImage
                )
            }
        val jobBackImage = backImage?.let {
            async {
                imageProcessingService.processSingleImage(
                    backImage
                )
            }
        }
        ProcessedImagePair(jobFrontImage.await(), jobBackImage?.await())
    }
}
```

Issues We Faced

- GraalVM
- MapStruct

GraalVM

- Issues with kotlin-reflect
- Requires a lot of reflection configuration

MapStruct

- Decorators not working when combined with Micronaut
- Inheritance differences when porting from Java

Conclusion

- We like it!
- Better code readability
- Easier code maintenance due to less code
- Compatibile with popular frameworks

Questions?

Thank You!